DB Name	Query	Hit Count	Set Name
USPT	(emulator$ with table$ with instruction with code$)	10	L2
USPT	(emulator$ with table$ with different with instruction with code$)	0	L1

(2) 5,751,982 interrupt

(5) 5,678,032 ~~interrupt~~

(9) 4,812,975 interrupt

translator

6,006,029

WEST

Your wildcard search against 2000 terms has yielded the results below

Search for additional matches among the next 2000 terms

Generate Collection

Search Results - Record(s) 1 through 10 of 10 returned.

☐ 1. Document ID: US 5781758 A

L2: Entry 1 of 10 File: USPT Jul 14, 1998

DOCUMENT-IDENTIFIER: US 5781758 A
TITLE: Software emulation system with reduced memory requirements

ABPL:
Memory requirements for an emulation system are reduced by generating semantic routines on demand during emulation, rather than statically storing all routines in the body of a software emulation system. The static portion of the emulator code that is loaded into the memory of the computer comprises only one copy of each different type of semantic routine. For the emulated instruction that corresponds to the one routine stored in the emulator code, a dispatch table entry comprises a pointer to the stored semantic routine. The dispatch table entries for the other emulated instructions of the same type comprise pointers to a semantic routine generator for instructions that have the same number of operands. This semantic routine generator locates the statically stored semantic routine and makes a copy of it, substituting the appropriate operands for the desired instruction in place of those in the statically stored routine. Once this modified copy of the static semantic routine has been generated and stored in memory, its address is entered into the dispatch table, in place of the pointer to the semantic routine generator. All subsequent calls to the new instruction are then emulated by using the dynamically generated semantic routine.

BSPR:
For the specific emulated instruction that corresponds to a semantic routine that is statically stored in the emulator code, the dispatch table entry comprises a pointer to the stored routine. The dispatch table entries for the other emulated instructions of the same type, which are not statically stored, contain pointers to a semantic routine generator for instructions that have the same number of operands. This semantic routine generator locates the statically stored semantic routine and makes a copy of it, substituting the appropriate operands for the desired instruction in place of those in the statically stored routine. Once this modified copy of the static semantic routine has been generated and stored in memory, its address is entered into the dispatch table, in place of the pointer to the semantic routine generator. All subsequent calls to the new instruction are then emulated by using the dynamically generated semantic routine.

Full	Title	Citation	Front	Review	Classification	Date	Reference	Claims	KWIC	Draw Desc	Image

☑ 2. Document ID: US 5751982 A

L2: Entry 2 of 10 File: USPT May 12, 1998

DOCUMENT-IDENTIFIER: US 5751982 A
TITLE: Software emulation system with dynamic translation of emulated instructions for increased processing speed

BSPR:
One characteristic of an emulation system which has a significant impact on its overall performance is the considerable execution time overhead which the emulation system imposes. In particular, a good percentage of the time required for emulation is spent in the dispatching operations. In general, each instruction generated by an application program, in the instruction set for the processor being emulated, causes the emulator to address the dispatch table, which results in a jump to the corresponding semantic routine in the ~~native, or~~ emulation, code. Thus, for each instruction in the emulated code, the following sequence of actions occurs: (a) fetching the instruction to be implemented, (b) ~~addressing the~~ dispatch table, (c) obtaining the pointer to the native code, (d) fetching the first instruction for the corresponding semantic routine in the native code, and (e) executing the semantic routine.

BSPR:
In the second phase, the selected code sequences are translated from the instruction set of the emulated processor into the instruction set of the emulating processor. For each emulated instruction in a selected code sequence, its equivalent code sequence in the native instruction set is obtained from the emulator's set of semantic routines, by indexing into the dispatch table with a binary code for the emulated instruction. The successively retrieved code sequences are cumulatively stored in an instruction buffer, until each instruction in the selected sequence has been translated.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc | Image

3. Document ID: US 5678032 A
L2: Entry 3 of 10 File: USPT Oct 14, 1997

DOCUMENT-IDENTIFIER: US 5678032 A
TITLE: Method of optimizing the execution of program instuctions by an emulator using a plurality of execution units

BSPR:
A standard simplified way of implementing an interpretative emulator is to employ a dispatch loop within the emulator to fetch an emulated program instruction from the emulated program instruction stream and use the binary value of the program instruction operation code as an address for indexing into a table in memory. The value of the table entry is the starting address of an emulation routine consisting of host instructions that implement the changes of state within the host system required to emulate the original program instruction. The dispatch loop causes a branch to the emulation routine whose instructions are then executed by the host system. The final host instruction within the emulation routine returns control back to the dispatch loop which then fetches the next emulated program instruction. It has been found that this process is time consuming.

DEPR:
Also, interpreter unit 72, as described in greater detail herein, includes routines which perform a dispatch function wherein the operation code of an emulated program instruction is used to index into an emulator jump table (not shown). The table contains a plurality of address pointer entries (e.g. up to 2.sup.16 entries). The pointer obtained by the indexing designates the set or RISC instructions of the routine or code fragment to be executed by the host system required to emulate the instruction of the original emulated program. According to the present invention, the interpreter unit 72 is organized in a manner which reduces or shortens the time required to perform the dispatch function (i.e., decreases the number of instructions contained in the dispatch loop).

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc | Image

---

☐ 4. Document ID: US 5668969 A

L2: Entry 4 of 10 File: USPT Sep 16, 1997

DOCUMENT-IDENTIFIER: US 5668969 A
TITLE: Address selective emulation routine pointer address mapping system

BSPR:
Interpretive emulation is the most desirable emulation technique in terms of emulation accuracy and robust performance; unfortunately, it is typically the slowest emulation technique. The most straightforward method of implementing an interpretive emulator is to employ a dispatch loop within the emulator to fetch a source instruction from the source program stream, and to use the binary value of the operation code within the source instruction to index a table in memory. The value of the table entry, referred to here as a "pointer," is the address of an emulation routine consisting of host instructions that implement the architectural changes of state required to emulate the original source instruction. The dispatch loop issues a jump to the address indicated by the pointer, after which the emulation routine is executed. The final host instruction within the emulation routine returns control to the dispatch loop, which fetches the next source instruction from the source program.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc | Image

5. Document ID: US 5574887 A

L2: Entry 5 of 10 File: USPT Nov 12, 1996

DOCUMENT-IDENTIFIER: US 5574887 A
TITLE: Apparatus and method for emulation routine pointer prefetch

BSPR:
Interpretive emulation is the most desirable emulation technique in terms of emulation accuracy and robust performance; unfortunately, it is typically the slowest emulation technique. The most straightforward method of implementing an interpretive emulator is to employ a dispatch loop within the emulator to fetch a source instruction from the source program stream, and to use the binary value of the operation code (opcode) within the source instruction to index a table in memory. The value of the table entry, referred to here as a "pointer," is the address of an emulation routine consisting of host instructions that implement the architectural changes of state required to emulate the original source instruction. The dispatch loop issues a jump to the address indicated by the pointer, after which the emulation routine is executed. The final host instruction within the emulation routine returns control to the dispatch loop, which fetches the next source instruction from the source program.

Full	Title	Citation	Front	Review	Classification	Date	Reference	Claims	KWIC	Draw Desc	Image

---

6. Document ID: US 5408622 A

L2: Entry 6 of 10 File: USPT Apr 18, 1995

DOCUMENT-IDENTIFIER: US 5408622 A
TITLE: Apparatus and method for emulation routine control transfer via host jump instruction creation and insertion

BSPR:
Interpretive emulation is the most desirable emulation technique in terms of emulation accuracy and robust performance; unfortunately, it is typically the slowest emulation technique. The most straightforward method of implementing an interpretive emulator is to employ a dispatch loop within the emulator to fetch a source instruction from the source program stream, and to use the binary value of the operation code within the source instruction to index a table in memory. The value of the table entry, referred to as a "pointer," is the address of an emulation routine consisting of host instructions that implement the architectural changes of state required to emulate the original source instruction. The dispatch loop issues a jump to the address indicated by the pointer, after which the emulation routine is executed. The final host instruction within the emulation routine returns control to the dispatch loop, which fetches the next source instruction from the source program.

Full	Title	Citation	Front	Review	Classification	Date	Reference	Claims	KWIC	Draw Desc	Image

---

7. Document ID: US 5392408 A

L2: Entry 7 of 10 File: USPT Feb 21, 1995

DOCUMENT-IDENTIFIER: US 5392408 A
TITLE: Address selective emulation routine pointer address mapping system

BSPR:
Interpretive emulation is the most desirable emulation technique in terms of emulation accuracy and robust performance; unfortunately, it is typically the slowest emulation technique. The most straightforward method of implementing an interpretive emulator is to employ a dispatch loop within the emulator to fetch a source instruction from the source program stream, and to use the binary value of the operation code within the source instruction to index a table in memory. The value of the table entry, referred to here as a "pointer," is the address of an emulation routine consisting of host instructions that implement the architectural changes of state required to emulate the original source instruction. The dispatch loop issues a jump to the address indicated by the pointer, after which the emulation routine is executed. The final host instruction within the emulation routine returns control to the dispatch loop, which fetches the next source instruction from the source program.

Full	Title	Citation	Front	Review	Classification	Date	Reference	Claims	KWIC	Draw. Desc	Image

---

☐ 8. Document ID: US 5361389 A

L2: Entry 8 of 10 File: USPT Nov 1, 1994

DOCUMENT-IDENTIFIER: US 5361389 A
TITLE: Apparatus and method for emulation routine instruction issue

BSPR:
Interpretive emulation is the most desirable emulation technique in terms of emulation accuracy and robust performance; unfortunately, it is typically the slowest emulation technique. The most straightforward method of implementing an interpretive emulator is to employ a dispatch loop within the emulator to fetch a source instruction from the source program stream, and to use the binary value of the operation code within the source instruction to index a table in memory. The value of the table entry, referred to as a "pointer," is the address of an emulation routine consisting of host instructions that implement the architectural changes of state required to emulate the original source instruction. The dispatch loop issues a jump to the address indicated by the pointer, after which the emulation routine is executed. The final host instruction within the emulation routine returns control to the dispatch loop, which fetches the next source instruction from the source program.

Full	Title	Citation	Front	Review	Classification	Date	Reference	Claims	KWIC	Draw. Desc	Image

---

☒ 9. Document ID: US 4812975 A

L2: Entry 9 of 10 File: USPT Mar 14, 1989

DOCUMENT-IDENTIFIER: US 4812975 A
TITLE: Emulation method

DEPR:
If the emulation program detects a supervisor call (SVC) instruction while sequentially executing machine language codes of the target machine, it references the area 414 in the local execution list (emulation mode control table) 41 established when the emulator is initiated (FIG. 4), and obtains one of the table entry, the first address of the SVC jump table 42 (steps 31 and 33). Then the microprogram references a value stored in a byte at an address corresponding to a combination of the obtained first address and the SVC code (0-255 indicating the operand value of SVC instruction) of the jump table 42. This value of SVC jump table 42 has been coded in advance by checking the SVC code in the target machine operating system 23 and contains (FF)16 for an entry corresponding to an SVC code which is associated with an input/output control instruction or a value other than (FF)16 for other entry. The emulation microprogram tests the entry value of the table 42. If the value is other than (FF)16 corresponding to an SVC code, it regards the instruction as an SVC instruction that can be processed on the target machine operating system 23 and carries out the emulation to perform SVC operation as predetermined in accordance with the target machine architecture (steps 34-35). If the entry value is (FF)16, the emulation microprogram regards the instruction as an SVC instruction which is associated with an input/output control macro instruction, saves the PSW indicating the current emulation mode state to the PSW save area 411 in the local execution list 41 depicted in FIG. 4, and at the same time, loads the new ECP interrupt PSW (412) indicating the entry address of the EPC 21 for translating the input/output control macro instruction of the target machine into the PSW, then releases the emulation mode and transfers control to the ECP 21 (steps 36-39).

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw. Desc | Image

☐ 10. Document ID: US 3955180 A

L2: Entry 10 of 10 File: USPT May 4, 1976

DOCUMENT-IDENTIFIER: US 3955180 A
TITLE: Table driven emulation system

DEPR:
The RWC table entry at location (77).sub.8 also has a counter code field of 1XXXXX which causes the emulator in processing a data transfer instruction, to generate a specification message as mentioned above. However, in the case of a control I/O instruction, the emulator forces a branch as explained herein. The RWC busy mask field in this table entry is used to indicate which read/write channels in the target system being emulated are currently assigned (i.e., busy). This field is updated by the ESP whenever a data transfer instruction is initiated or terminated. As explained herein, the emulator accesses but never alters the contents of this field when it is performing a RWC busy test with a legal RWC code not (00).sub.8 or (77).sub.8 for a data transfer instruction or control instruction.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw. Desc | Image

Generate Collection

Term	Documents
EMULATOR$	0
EMULATOR.USPT.	1992
EMULATORBEDIENEINHEIT.USPT.	1
EMULATORFOR.USPT.	2
EMULATORS.USPT.	578
EMULATORSTATEPTR.USPT.	2
EMULATORS-SWD-A.USPT.	1
EMULATORS:.USPT.	3
EMULATORY.USPT.	1
EMULATORZENTRALEINHEIT.USPT.	1
((EMULATOR$ WITH TABLE$ WITH INSTRUCTION WITH CODE$)).USPT.	10

There are more results than shown above. Click here to view the entire set.

Display | 10 | Documents, starting with Document: | 10

Display Format: KWIC | Change Format

WEST

#7 interrupt

Generate Collection

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 5751982 A

L7: Entry 1 of 1 File: USPT May 12, 1998

DOCUMENT-IDENTIFIER: US 5751982 A
TITLE: Software emulation system with dynamic translation of emulated instructions for increased processing speed

DEPR:
In accordance with the present invention, this processing overhead can be considerably reduced through dynamic translation of selected code blocks in the emulated application program. To implement this feature, the code blocks which are emulated frequently enough to warrant dynamic translation are first identified. This can be carried out by recording program counter values that produce non-sequential changes in the sequence of instructions being emulated. Whenever an instruction from the CISC code is emulated that results in a non-sequential change to the program counter, the new program counter value is recorded, for example by pushing its value onto a programmatic stack. In essence, each recorded program counter value represents the starting point of a new code block. Whenever there is a break in the operation of the emulator, for example as the processor stops emulation to service a special event or an interruption, the accumulated values are removed from the stack and analyzed to identify code blocks that are emulated more than a defined number of times within a predetermined time window. For example, if a particular block is emulated more than 256 times within a period of about 16 milliseconds, it may be selected. Any suitable approach can be employed to select the program counter values that identify the code blocks which occur with sufficient frequency. In the preferred embodiment of the invention, as each recorded program counter value is removed from the stack, its value is used as a hash index into a table of frequency counts, and the corresponding entry in the table is incremented by one. When the count is incremented beyond a predetermined threshold value, the program counter value corresponding to that table entry is placed on a list of code blocks to be dynamically translated.

DEPR:
This entire dynamic translation procedure, namely (a) the analysis of program counter values, (b) the identification of frequently emulated code blocks, and (c) the translation and storage of selected code blocks in the buffer, preferably takes place during the interruption of the emulation, i.e. prior to the time that the event which interrupted the emulation is serviced.

CLPR:
5. The method of claim 4 wherein said counting step comprises the steps of loading a value associated with said non-sequential change in a stack upon each detected occurrence, emptying said stack in response to an interruption in an emulation operation, and counting the number of occurrences of each value in said stack.

Full	Title	Citation	Front	Review	Classification	Date	Reference	Claims	KWIC	Draw. Desc	Image

Generate Collection

wtoms

WEST

#3

Generate Collection

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 4812975 A

L8: Entry 1 of 1 File: USPT Mar 14, 1989

DOCUMENT-IDENTIFIER: US 4812975 A
TITLE: Emulation method

ABPL:
A method for emulating programs in a system includes a plurality of first and second data processors having different instruction word sets. An instruction which interrupts the operating system on the first data processor is defined. When the instruction is detected in a program running on the first data processor, it is determined whether or not the instruction is an instruction associated with an input/output macro instruction. If it is found, as a result of the determination, that this is the case, an interrupt is caused in a program running on the second data processor which controls the emulation, and the input/output macro instruction output from an emulated program is translated into an input/output macro instuction for the operating system, thereby implementing an emulation with a minimized overhead.

BSPR:
The present invention is materialized in a system comprising data processors, for example, a first data processor and a second data processor which have a different instruction word set, respectively. It is assumed that the user programs in the first data processor are processed under control of a first operating system and that those in the second data processor are processed under control of a second operating system. The programs in the first data processor are assigned as the programs to be emulated (to be referred to as emulated programs hereafter) and the specific instructions are defined in these emulated programs. Such an instruction like a supervisor call instruction, is one that causes an interrupt in the first operating system. When.. the interrupt instruction is detected, it is examined to determine whether or not this is an interrupt instruction associated with an input/output control macro instruction. If it is found, as the result of the examination, that this is the case, an interrupt occurs in a program on the second data processor which controls the emulation. Then, the input/output control macro instructions issued by the emulated program are translated into the corresponding input/output control macro instructions for the second operating system.

DEPR:
The EXL instruction format consists of an instruction code part (EXL), a B.sub.2 part comprising four bits for indicating a general-purpose register to be assigned as the base register, and a D.sub.2 part comprising 12 bits which indicate a binary value. When an EXL instruction is issued, the second operand address is register specified by the B.sub.2 part to the D.sub.2 part. The second operand address indicates an address on the main storage device at which a local execution list 41 (FIG. 4) is stored. The local execution list 41 comprises an area 411 for saving the emulation mode PSW necessary for the emulator operation, an area 412 for storing therein the new PSW for the ECP interrupt, a general-purpose register save area 413, an area for storing therein the first address of the jump table for the superviser call (SVC) instruction, and other areas. When the EXL instruction is executed, the pertinent values are set to the PSW, general-purpose register, etc. in the local execution list 41, then the system enters the local execution (emulator operation) mode.

DEPR:

In accordance with the emulation method of the present invention, since an input/output control instruction specified by the user program 24 in the emulated program 22 is issued in the form of an input/output macro instruction of the operating system 23 in the emulated program 22, and it is ordinarily an interrupt instruction, such as an SVC instruction, to the operating system 23; the input/output instruction is emulated at the macro instruction level rather than at the machine language level which has been adopted in the conventional emulation method.

DEPR:
In accordance with the present invention, when an SVC instruction is detected during the execution of the emulated program 22, the micro program for executing the emulation searches the entries of the local execution list 41 (FIG. 4) for the SVC jump table first address entry 414. Each entry of the table 42 is referenced by use of the entry 414 stored in the local execution list 41. Then, the instruction is examined to determine whether or not it is a supervisor call instruction associated with an input/output control macro instruction. If it is found, as the result of the examination, to be a supervisor call instruction for an input/output control instruction, the operating system 23 for the emulated program 22 is not interrupted and the emulation mode is released, then an interrupt is caused in the interface program, that is, ECP 21 which operates in the native mode. The ECP 21 analyses and translates the input/output control instruction issued from the emulated program 22 into an input/output control macro instruction for the native mode operating system 10, then issues the obtained instruction.

DEPR:
If the emulation program detects a supervisor call (SVC) instruction while sequentially executing machine language codes of the target machine, it references the area 414 in the local execution list (emulation mode control table) 41 established when the emulator is initiated (FIG. 4), and obtains one of the table entry, the first address of the SVC jump table 42 (steps 31 and 33). Then the microprogram references a value stored in a byte at an address corresponding to a combination of the obtained first address and the SVC code (0-255 indicating the operand value of SVC instruction) of the jump table 42. This value of SVC jump table 42 has been coded in advance by checking the SVC code in the target machine operating system 23 and contains (FF)16 for an entry corresponding to an SVC code which is associated with an input/output control instruction or a value other than (FF)16 for other entry. The emulation microprogram tests the entry value of the table 42. If the value is other than (FF)16 corresponding to an SVC code, it regards the instruction as an SVC instruction that can be processed on the target machine operating system 23 and carries out the emulation to perform SVC operation as predetermined in accordance with the target machine architecture (steps 34-35). If the entry value is (FF)16, the emulation microprogram regards the instruction as an SVC instruction which is associated with an input/output control macro instruction, saves the PSW indicating the current emulation mode state to the PSW save area 411 in the local execution list 41 depicted in FIG. 4, and at the same time, loads the new ECP interrupt PSW (412) indicating the entry address of the EPC 21 for translating the input/output control macro instruction of the target machine into the PSW, then releases the emulation mode and transfers control to the ECP 21 (steps 36-39).

CLPV:
a. detecting an interrupt instruction defined in a first program in which an instruction of said target machine is executed;

CLPV:
b. determining whether or not said interrupt instruction is an input/output control macro instruction by use of an operand value of said interrupt instruction; and

CLPV:
c. if said interrupt instruction is determined to be an instruction associated with an input/output control macro instruction in said step b, bypassing the operation system of said target machine by translating said input/output control macro instruction from said first program into an input/output control macro instruction for a second program, in which an instruction of said native machine is executed, under control of a program for controlling said emulation.

WEST

Generate Collection

L2: Entry 1 of 15 File: USPT Dec 21, 1999

DOCUMENT-IDENTIFIER: US 6006029 A
TITLE: Emulating disk drives of a first system on a second system

ABPL:
The emulation of a first system disk drive on a second processing system including a second system user level process including first system user and executive tasks issuing disk input/output requests. An emulator level is interposed between the second system user level process and a kernel level and includes a pseudo device driver corresponding to the first system disk drive and the kernel level includes a kernel process corresponding to the pseudo device driver and emulating the disk drive. The pseudo device driver and the kernel process execute in a second system process to emulate the operations of the disk drive and the kernel process emulating the disk drive is a file input/output process. The pseudo device driver includes a pseudo device queue, a return queue and a queue manager responsive to first system disk input/output instructions and to completed disk operations. The second system includes a resource control table containing a disk drive type identification as a SCSI type drive and the kernel process reads the file capacity of the second system file emulating the first system disk drive and provides the file capacity to the requesting task as the disk drive capacity.

DRPR:
FIG. 8 is the address translation mechanism and memory space mapping mechanism of the emulation mechanism.

DEPR:
5. Addresses and Address Translation

DEPR:
It will be noted, as described previously, that Software Active Queue (SAQ) 88, the Pseudo Device Queues (PSDQs) 86, and INTERPRETER 72 are provided to emulate the corresponding mechanisms of First System 10, that is, First System 10's input/output devices and central processing unit, as seen by Executive Program Tasks (EXP Tasks) 28 and Tasks 30. As such, Executive Program Tasks (EXP Tasks) 28 and Tasks 30 will provide memory addresses to the Pseudo Device Queues (PSDQs) 82 and INTERPRETER 72 according to the requirements of the native memory access and management mechanisms of First System 10 and will expect to receive memory addresses from Software Active Queue (SAQ) 88 and INTERPRETER 72 in the same form. Second System Kernel Processes (SKPs) 66, Lower Communications Facilities Layer Processes (LCFLPs) 78, the hardware elements of Second System 54 and other processes executing as native processes in Second System 54, however, operate according to the memory addressing mechanisms native to Second System 54. As such, address translation is required when passing requests and returning requests between Emulator Executive Level (EEXL) 68 and Second System Kernel Level (SKernel) 64.

DEPR:
As described, INTEPRETER 70 is provided to interpret First System 10 instructions into functionally equivalent Second Second 54 instructions, or sequences of instructions, including instructions pertaining to memory operations. As such, the address translation mechanism is also associated with INTERPRETER 72, or is implemented as a part of INTERPRETER 72, and is indicated in FIG. 3 as Address Translation (ADDRXLT) 98 and will be described in detail in a following discussion.

DEPR:
As described above with reference to FIGS. 2 and 3, the First System 10 tasks and programs executing on Second System 54, Second System 54's native processes and mechanisms and the Second System 54 mechanisms emulating First System 10

mechanisms share and cooperatively use Second System 54's memory space in Second System Memory 58b. As a consequence, it is necessary for Second System 54, the First System 10 tasks and programs executing on Second System 54, and the emulation mechanisms to share memory use, management, and protection functions in a manner that is compatible with both Second System 54's normal memory operations and with First System 10's emulated memory operations. The emulation of First System 10 memory operations in Second System 54 in turn requires emulation of First System 10's memory management unit, that is, First System 10's hardware and software elements involved in memory space allocation, virtual to physical address translation, and memory protection in Second System 54. As described below, this emulation is implemented through use of Second System 52's native memory management unit to avoid the performance penalties incurred through a complete software emulation of First System 10's memory management unit.

DEPR:
As is well known, most systems operate upon the basis of virtual addresses and perform virtual to physical address translations relative to a predetermined base address, that is, by adding a virtual address as an offset address to the base address to determine the corresponding address in physical address space of the system. While First System 10 and Second System 52 may both use such addressing schemes, the actual addressing mechanisms of the two system may differ substantially, as may the memory protection schemes.

DEPR:
Referring to FIG. 8, and to FIGS. 2, 3, 5 and 6, therein is illustrated the mechanisms implemented on Second System 54 to emulate the memory access, protection, and management mechanisms of First System 10. It must be recognized in the following that the emulation of First System 10 memory operations on Second System 54 involves two differerent address conversion operations, one being the conversion of First System Virtual Addresses (FSVAs) 126 done by INTERPRETER 72 and the second being the conversion of First System Virtual Addresses (FSVAs) 126 done by Pseudo Device Drivers (PSDDs) 74. Each of these conversions is accomplished through translation and through mapping of First System 10 system and user memory areas into Second System 54 segments. The following will first describe the address translation operation performed by INTERPRETER 72, and then will describe the address translation operation performed by Pseudo Device Drivers (PSDDs) 74.

DEPR:
First considering the process of INTERPRETER 72 address translation, as has been described above, each First System Virtual Address (FSVA) 126 is comprised of a Most Significant Bits field (MSB) 128 and an Address field (ADDR) 130 wherein Most Sigificant Bits field (MSB) 128 contains a bit field whose value identifies whether the address is directed to an executive memory area, that is, System Memory (SYSMEM) 110 area, or to an Independent-Memory Pool (IPOOL) 112. For example, the Most Sigificant Bits field (MSB) 128 may contain the value 0000 (0) when the request is directed to the System Memory (SYSMEM) 110 area and the value 0001 (1) when the request is directed to an Independent-Memory Pool (IPOOL) 112 area.

DEPR:
As indicated in FIG. 8, the First System Virtual Address (FSVA) 126 of a request which includes a memory access is provided to Address Translation (ADDRXLT) 98. Address Translation (ADDRXLT) 98 includes a Word To Byte Shifter (WBS) 148 which performs an initial translation of the First System Virtual Address (FSVA) 126 from the First System 10 format, in which addresses are on a per word basis, to a Second System 54 virtual address, in which addresses are on a per byte basis. This translation is performed by a left shift of the First System Virtual Address (FSVA) 126 and, in the translation and as indicated in FIG. 7, the value in the Most Sigificant Bits field (MSB) 128 field of the First System Virtual Address (FSVA) 126 is transformed from 0000 (0) or 0001 (1) to 0000 (0) or 0010 (2), respectively.

DEPR:
Having performed the translation of a First System Virtual Address (FSVA) 126 into a per byte address, Address Translation (ADDRXLT) 98's Ring Adder (RNGA) 150 will read a System Status Register (SSR) 152 which, among other information, contains a Ring Number (RNG) 154 which contains a value indicating the First System 10 ring in which the task is executing, that is, a value of 0, 1, 2 or 3.

As described, Ring 0 is reverved for system operations while Rings 1, 2 and 3 are used for user tasks. If the task is executing in Ring 0, that is, in system space, Ring Adder (RNGA) 150 will add 3 to the value (0 or 2) contained in Most Significant Bits field (MSB) 128 of the shifted First System Virtual Address (FSVA) 126. If the task is not executing in Ring 0, that is, is executing in Rings 1, 2, or 3 and thus in user task space, Ring Adder (RNGA) 150 will add 4 to the value (0 or 2) contained in Most Significant Bits field (MSB) 128 of the shifted First System Virtual Address (FSVA) 126. The final result will be a byte oriented First System Virtual Address (FSVA) 126 having a Most Significant Bits field (MSB) 128 which contains a value of 3, 4, 5 or 6, thereby indicating the Second System 54 memory space segment in which the address lies and an Address (ADDR) field 130 identifying a location within the segment.

DEPR:
As has been described, each Pseudo Device Driver Queue (PSDQ) 86 is associated with a corresponding Second System Kernel Process (SKP) 66 which executes the requests in the Pseudo Device Driver Queue (PSDQ) 86 and any Pseudo Device Driver Queue (PSDQ) 86 may contain requests from a plurality of tasks, each task in turn being associated with and executed in an Independent-Memory Pool (IPOOL) 112 area which is mapped into a Second System 54 memory segment by address translator (ADDRXLP) 96 which includes a Server Pool Descriptor Linked Set (SPDLS) associated with the Pseudo Device Driver Queue (PSDQ) 86, Task Control Block (TCB) 32, Segment Descriptor Table 156, and Memory Pool Array 162.

DEPR:
It may be seen from the above descriptions, therefore, that, for any first system virtual address generated by a First System 10 task executing on Second System 54, INTERPRETER 72 will translate the First System 10 virtual address into a byte oriented virtual address containing a virtual address location within a segment and identifying a Segment 3, 4, 5 or 6 containing the location. The INTERPRETER 72 mapping of segments via ADDRXLT98 will in turn map each segment identified by an address translation into an Independent Memory Pool Identification (IPOOLID) 160 for the current task. The Segment/Independent Memory Pool mapping mechanism (i.e., ADDRXLP96) of the Pseudo Device Driver (PSDD) 74 executing the task request associated with the First System 10 virtual address will map the segment identified by the address translation mechanism to a current Independent Memory Pool (IPOOL) 112 location in System 54's memory by providing the base address corresponding to the Independent Memory Pool Identification (IPOOLID) 160.